Page Object Model

1. Page Object Model

Page Object Model (POM) is a design pattern, popularly used in test automation that creates Object Repository for Web UI elements. The advantage of the model is that it reduces code duplication and improves maintenance and increase readability.

Under this model, for each web page in the application, there should be a corresponding Page Class. This Page class will identify the WebElements of that web page and also contains Page methods which perform operations on those WebElements.

1. Page Object Model:

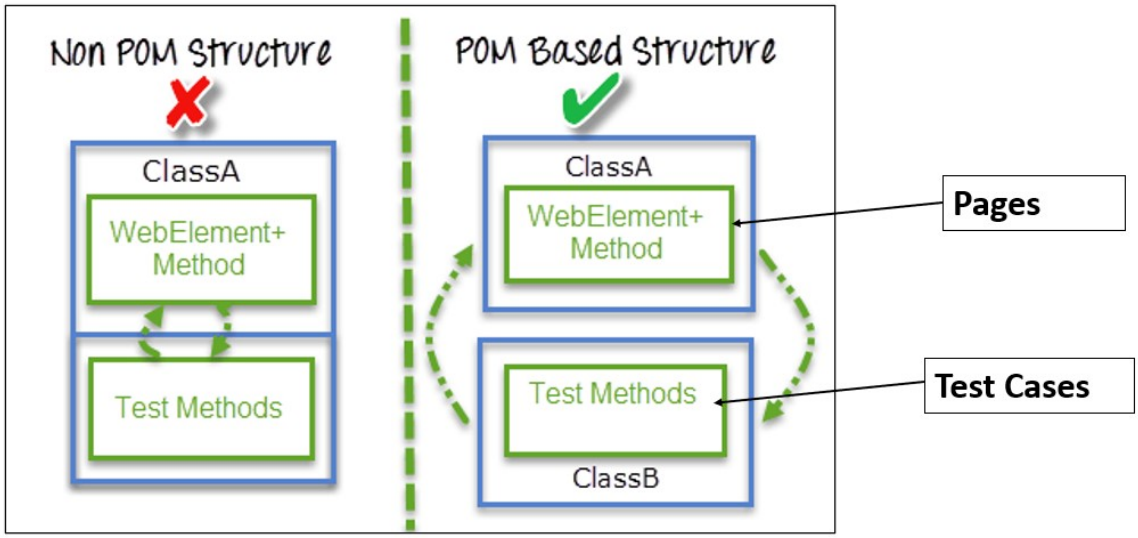
1. Create Object Repository for WebUI Elements
2. Create objects for pages through which we perform operation via TestNG file through @Test method

2. Advantages:

1. Reduces Code Duplication – Maintainability
2. Increase Code Readability - Readability
3. Increase Code Reusability

3. Separate Pages and Tests

1. Page Object Design Pattern says operations in the UI should be separated from test. This concept makes our code cleaner and easy to understand



1. Test Scenarios in Flipkart

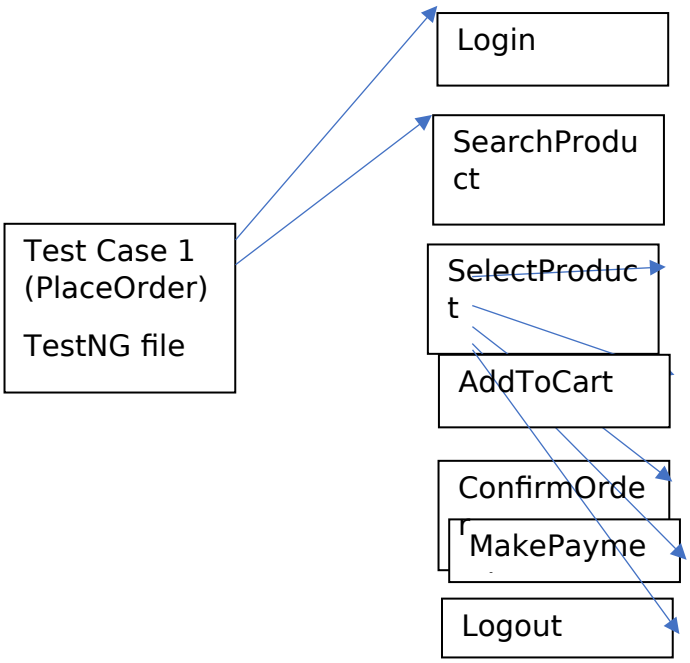
Test Case 1: PlaceOrder **(**1st @Test method**)**

Test Case 2: AddtoCart (2nd @Test method)

Test Case 3: AddDeliveryAddress (3rd @Test method)

2. Scenario Navigation:

Login -> Search a Product -> Select a Product -> Click Add to Cart button -> Place Order -> Make a Payment -> Logout



3. Website: Flipkart

Pages involved in the <u>PlaceOrder</u> Scenario:

1. **HomePage**
2. **SearchProductPage**
3. **SearchResultPage**
4. **ProductDetailPage**
5. **PlaceOrderPage**
6. **MakePaymentPage**
7. **LogoutPage**

4. Class Name: FlipkartTest

@BeforeClass() / @BeforeMethod → launch browser

Test Case 1: testPlaceOrder() - @Test Method contains following steps

1. Step 1 - login()
2. Step 2 - searchProduct() ->
3. Step 3 - selectProduct()
4. Step 4 - addToCart()
5. Step 5 - confirmOrder()
6. Step 6 - makePayment()
7. Step 7 - logout()

@AfterClass() / @AfterMethod → close browser

5. Test Case: PlaceOrderTest

TestNG File	Page Class	Method in Page Classs	
BeforeClass() / Before Method()	To initialize the browser and launch the website		
testPlaceOrder() - **@Test method**			
Login step	HomePage	login()	Return the operation status to corresponding method
searchProduct step	SearchProductPage	searchProduct()	
selectProduct step	SearchResultPage	selectProduct()	
addToCart step	ProductDetailPage	addToCart()	
confirmOrder step	PlaceOrderPage	confirmOrder()	
makePayment step	MakePaymentPage	makePayment()	
logout step	LogoutPage	logout()	

AfterClass() / AfterMethod	To close the browser

6. Create Object Repository

driver.findElement(By.id(“username1”)).sendKeys(“standard_user”);

username2

1000 times

By userName = By.id(“username2”);
driver.findElement(username).sendKeys(“standard_user”);

driver.findElement(By.xpath("//span[text()='Products']")).getText();

By productText = By.xpath("//span[text()='Products']");

driver.findElement(productText).getText();

S.No	TestNG @Test Method	Page Package	Object Repository + Methods at Page file
	BeforeClass()		
1	Step 1 - login	**HomePage**	**validateLogin()**
2	Step 2 - searchProduct	**SearchProductPage**	**validateSearchProduct()**
3	Step 3 - selectProduct	**SearchResultPage**	**validateSelectProduct()**
4	Step 4 - addToCart	**ProductDetailPa**	**validateAddToCart**

		ge	**()**
5	Step 5 - confirmOrder	**PlaceOrderPage**	**validatePlaceOrder()**
6	Step 6 - makePayment	**MakePaymentPage**	**ValidateMakePayment()**
7	Step 7 - logout	**LogoutPage**	**ValidateLogout()**
	AfterClass()		

Scenario 2: AddProductToCart

Navigation Flow / Traversal Flow: (Below are @Test methods or Test Cases)

1. login step
2. searchProduct step
3. selectProduct step
4. addtToCart step
5. checkProductsInCartpage step
6. logout step

Pages:

1. LoginPage
2. SearchProductPage
3. SearchResultPage
4. AddtoCartPage
5. CartDetailsPage
6. LogoutPage

Scenario: AddToCart

S.No	TestNG @Test Method	Page class	Object Repository +

			Methods at Page file
	BeforeClass()		
1	**login step**	**HomePage**	**validateLogin()**
2	**seachProduct step**	**SearchProduct Page**	**validateSearchProduct()**
3	**selectProduct step**	**SearchResultPage**	**validateSelectProduct()**
4	**addToCart step**	**ProductDetailPage**	**validateAddToCart()**
5	**checkProductsInCartpage step**	**CartPage**	**validateProductsInCart()**
6	**logout step**	**LogoutPage**	**ValidateLogout()**
	AfterClass()		

Scenario 3: AddDeliveryAddress

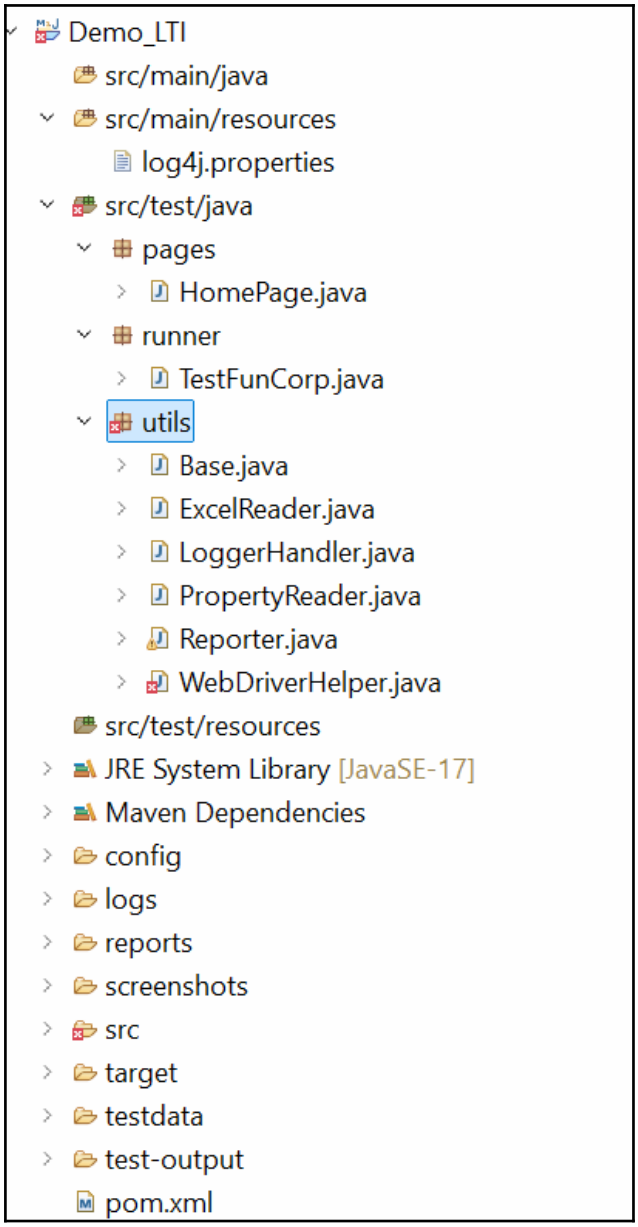
Navigation Flow / Traversal Flow:

1. loginTest()
2. clickMyaccount()
3. addDeliveryAddressTest()
4. logoutTest()

Pages:

1. LoginPage
2. HomePage
3. AddressPage
4. LogoutPage

2. Project Structure in POM



Ignore below one

src/main/java

com.flipkart.pages → package

- LoginPage

- SearchProductPage
- SelectProductPage

com.flipkart.base

- TestBase (contains reusability methods such as launchBrower(), navigateToURL(), captureScreenshot(), wait())

com.flipkart.utils

- ExcelUtil.java

src/main/resources

Config

config.properties

src/test/java

com.flikart.testscenairos

1. PlaceOrder.java
2. AddTiCart.java
3. AddDeliveryAddress

src/test/resources

TestData

Screenshots

Package

Folder

3. Step by Step procedure in POM Project

1. Create Maven Project
2. Create the above Packages (yellow) and Folders (Green) under the appropriate sections

3. Create TestNG file(PlaceOrder) under com.amazon.testscenarios which is under src/test/java
4. Create as many @Test methods as needed for your scenario
5. Now you will get TestNG related error in the PlaceOrder file, if so,
    a. google for “maven repository for TestNG”
    b. Open the mvnrepository.com site
    c. Select the appropriate TestNG version
    d. Copy the code under Maven Tab
    e. Add the below dependencies tag between <version> and <project> tag Then paste it in the pom.xml file

    <dependancies>

    .....

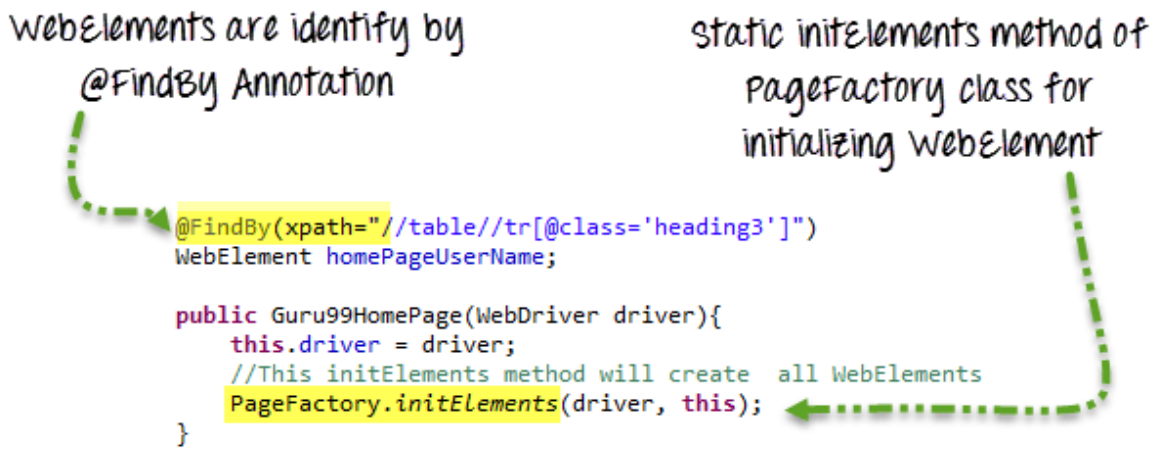
    ......

    </dependancies>

    f. Save the file and observe TestNG related jar files downloaded in “Maven Dependencies” folder
6. Create TestBase class under “com.amazon.base” package and place necessary maven repository in the pom.xml file
7. Extend the TestBase class in the PlaceOrder class
8. In the BeforeClass() method of PlaceOrder class use below method
    a. launchBrowser()
    b. navigateToURL()
9. Create the code for login functionality under loginTest() method
10. Create LoginPage class under “com.amazon.pages” package
11. Create object repositories for UI Elements to interact with
12. Create validateLogin() method which perform login operaion
13. Use Explicit Wait to validate the result
14. Return the result to loginTest() method
15. Develop the other @Test method same as above

4. PageFactory

Page Factory in Selenium is an inbuilt Page Object Model framework concept for Selenium WebDriver but it is very optimized. It is used for initialization of Page objects or to instantiate the Page object itself. It is also used to initialize Page class elements without using "FindElement/s."



https://www.browserstack.com/guide/page-object-model-in-selenium

```
@FindBy(xpath = "//h1")
WebElement Header;
@FindBy(xpath = "//*[@id='signupModalButton']")
WebElement getStarted;
public BrowserStackHomePage(WebDriver driver) {
```

```
this.driver = driver;
PageFactory.initElements(driver, this);
}
public void veryHeader() {
String getheadertext = Header.getText();
assertEquals("App & Browser Testing Made Easy", getheadertext);
}
public void clickOnGetStarted() {
getStarted.click();
}
```

```
public class BrowserStackSignUpPage {
WebDriver driver;
@FindBy(xpath = "//h1")
WebElement Header;
@FindBy(xpath = "//*[@id='user_full_name']")
WebElement userName;
@FindBy(xpath = "//*[@id='user_email_login']")
WebElement businessEmail;
@FindBy(xpath = "//*[@id='user_password']")
WebElement password;
public BrowserStackSignUpPage(WebDriver driver) {
this.driver = driver;
PageFactory.initElements(driver, this);
}
public void veryHeader() {
String getheadertext = Header.getText().trim();
assertEquals("Create a FREE Account", getheadertext);
}
public void enterFullName(String arg1) {
userName.sendKeys(arg1);
}
public void enterBusinessEmail(String arg1) {
businessEmail.sendKeys(arg1);
}
public void enterPasswrod(String arg1) {
password.sendKeys(arg1);
}
}
```

```
public class BrowserStackSetup {
String driverPath = "C:\\geckodriver.exe";
WebDriver driver;
BrowserStackHomePage objBrowserStackHomePage;
BrowserStackSignUpPage objBrowserStackSignUpPage;
@BeforeTest
public void setup() {
System.setProperty("webdriver.chrome.driver", "C:\\BrowserStack\\
chromedriver.exe");
driver = new ChromeDriver();
```

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("https://www.browserstack.com/");
}
@Test(priority = 1)
public void navigate_to_homepage_click_on_getstarted() {
objBrowserStackHomePage = new BrowserStackHomePage(driver);
objBrowserStackHomePage.veryHeader();
objBrowserStackHomePage.clickOnGetStarted();
}
@Test(priority = 2)
public void enter_userDetails() {
objBrowserStackSignUpPage = new BrowserStackSignUpPage(driver);
objBrowserStackSignUpPage.veryHeader();
objBrowserStackSignUpPage.enterFullName("TestUser");
objBrowserStackSignUpPage.enterBusinessEmail("TestUser@gmail.com");
objBrowserStackSignUpPage.enterPasswrod("TestUserPassword");
}
}
```

5. Difference Between Page Object Model and Page Factory in Selenium

Page Object Model	Page Factory
Finding web elements using **By**	Finding web elements using **@FindBy**
POM does not provide lazy initialization	Page Factory does provide lazy initialization
Page Object Model is a design pattern	PageFactory is a class which provides implementation of Page Object Model design pattern

In POM, one needs to initialize every page object individually	In PageFactory, all page objects are initialized by using the initElements() method

Facebook New Account Creation:

CreateNewAccount.java

driver.findElements(By.name(“firstname1”)).sendKeys(“Senthil”) – 10 places

driver.findElements(By.name(“lastname”)).sendKeys(“Nata”)

driver.findElement(By.id(“uname”)).sendKeys(“Senthil”); - User Name text field

driver.findElement(sendKeys(“adfsd”);   - Password text field

driver.findElement(By.className(“abcded”)).click(); - Login button

By userNameLocator = By.id(“uname1”);

By pwdLocator = By.id(“pwd”)

1000 places - driver.findElement(userNameLocator).sendKeys(“senthil”);

1000 places - driver.findElement(pwdLocator).sendKeys(“adsdfdf”);

By firstNameLocator = By.name(“firstname1”);

By passwordLocator = By.xpath(“//*[@id="password1_step_input"]”);

driver.findElements(firstNameLocator).sendKeys(“Senthil”) – 10 places

driver.fineElements(passwordLocator).sendKeys(“abcd”); - 4 places

Test:

PlaceOrder.java(testNG file)

```
@Test
Public void launchHomepage(){
        LoginPage loginPage = new LoginPage();
        loginPage.validateLogin();

}
```

LoginPage.java Loginpage() – 3 - locators for login – LaunchPage.java

SearchProductPage Searchproduct() – 1 -locator for search text box

Productdetail() – 1 -locator for click product

MakePayment() – 1 locator

Orderconfirmm() – 10 - many locators for address fields

Logout() – locator for logout button

Reference for POM Code

https://www.perfecto.io/blog/page-object-model-in-selenium

1. TestNG file
2. Page files
3. Excel Files
4. Property files
5. Drivers
6. Test data
7. Screenshots

Steps to develope POM Based code:

1. Create TextNG file for scenario1
2. Execute @BeforeClass
    a. TestNG class will extends TestBase class which has WebDriver, methods for implementations of launching the browser, navigating to URL, taking screenshot, reading from Property files.
3. Develop code for validateLoginTest() @Test method in TestNG file.
    a. Create necessary Page file and Object Repository and methods to perform the operation
    b. After the operation, perform validation and send the validation to back to @Test method
    c. Assert in the @Test method
4. Repeat the same step 3 and 4 for each @Test method